

**REMARKS**

Favorable reconsideration of this application is requested in view of the following remarks. Claims 1-10 are pending in the application. Reconsideration of the claim is respectfully requested.

In paragraph 1 on page 2 of the Office Action, the Office Action stated that the information disclosure statement filed 9/28/04 fails to comply with 37 CFR 1.98(a)(2), which requires a legible copy of each cited foreign patent document. Applicants have attached a legible copy of the Scholl reference hereto and believe they are now in compliance 37 CFR 1.98(a)(2).

In paragraph 3 on page 2 of the Office Action, claims 1-10 were finally rejected under 35 USC §102(e) as being anticipated by Niamir (US Patent Pub. 2002/0027567). Applicants respectfully traverse the rejections.

First, Niamir fails to teach or suggest at least a server having pre-authorization for automatic access to at least one digital image file on a user computer and for permitting access to at least one digital image file at said server by a third party. Rather, Niamir discloses a peer-to-peer computer system in which files are transferred between user computers in a peer-to-peer manner bypassing a central server. *See* Abstract and [0005]. In the peer-to-peer system of Niamir, users who are authenticated by an authentication server 64 can then have their listings 30 saved at a central search server (CSS) 16 from where the listings 30 can be retrieved by other users of system 10. *See* [0099].


In sharp contrast, Applicants' invention requires a server for allowing controlled access to a digital image file of an image stored on a user computer. The server has pre-authorization for automatic access to at least one digital image file on a user computer and for permitting access to said at least one digital image file at the server by a third party. That is, the server has pre-authorization for automatic access to at least one digital image file on a user computer and the server permits a third party to access the digital image files *at the server*. Accordingly, Applicants respectfully disagree with the Office Action's assertion in the *Response to Arguments* that Applicants' claims do not disclose that the server is the only party that has direct access with the user computer.

Second, Niamir fails to teach or suggest at least a server monitoring access to a digital image file by a third party whenever access by said third party to said digital image file is done. Rather, Niamey merely discloses that users can cause certain searches to run automatically to monitor new listings 30 and to alert the user when new listings match the user's search criteria. *See* [0074]. However, Niamir does not disclose monitoring access to said digital image file by said third party.

Therefore, in view of the above remarks, Applicants' independent claims 1, 6 and 7 are patentable over the cited reference. Because claims 2-5, 10 and 8-9 depend from claims 1 and 7, and include the features recited in the independent claims, Applicants respectfully submit that claims 2-5 and 8-10 are also patentably distinct over the cited reference. Nevertheless, Applicants are not conceding the correctness of the Office Action's rejection with respect to such dependent claims and reserve the right to make additional arguments if necessary.

In view of the foregoing it is respectfully submitted that the claims in their present form are in condition for allowance and such action is respectfully requested.

Respectfully submitted,

  
\_\_\_\_\_  
Attorney for Applicant(s)  
Registration No. 53,950

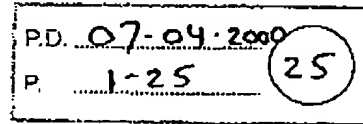
Thomas J. Strouse/phw  
Rochester, NY 14650  
Telephone: 585-588-2728  
Facsimile: 585-477-4646

If the Examiner is unable to reach the Applicant(s) Attorney at the telephone number provided, the Examiner is requested to communicate with Eastman Kodak Company Patent Operations at (585) 477-4656.

XP-002249287

napster messages  
 =====

by drscholl@users.sourceforge.net  
 April 7, 2000



## 0. Foreward

This is meant to be an open specification. If you find errors or know of additional functionality not described hereafter, please send me email. It benefits the entire community to have a complete and accurate protocol specification. Not only does it allow for clients to be developed for any platform, but also decreases the strain on the server having to parse out bad client messages.

Disclaimer: The following information was gathered by analyzing the protocol between the linux nap client and may not resemble the official Windows client protocol.

## 1. Recent Changes

- \* Section 3: 430-432 channel invite messages (2001-04-07)
- \* Section 3: WMA-FILE used as checksum for .wma files (2001-03-12)
- \* numeric 607 adds download client's speed (2000-12-28)
- \* added new numerics 640-642 for direct client to client browsing support (2000-12-23)

## 2. Client-Server protocol

Napster uses TCP for client to server communication. Typically the servers run on ports 8888 and 7777. Note that this is different from the 'metaserver' (or redirector) which runs on port 8875.

each message to/from the server is in the form of  
 <length><type><data>

where <length> and <type> are 2 bytes each. <length> specifies the length in bytes of the <data> portion of the message. Be aware that <length> and <type> appear to be in little-endian format (least significant byte goes first). For example, in the C language you would encode the number 1 as

```
const unsigned char num[2] = { 0x01, 0x00 };
```

and 256 would be encoded as

```
const unsigned char num[2] = { 0x00, 0x01 };
```

(The above is for illustrative purposes only, there are much quicker ways to actually encode a number. -ed)

The <data> portion of the message is a plain ASCII string.

Note that in many cases, strings are passed as double-quoted entries. For example, filenames and client id strings are always sent as

```
"random band - generic cowboy song.mp3"
```

or

```
"nap v0.8"
```

Where required, double quotes are used in the description of the messages below.

Some additional information about use of quotes inside of quotes:

- > The answer is, no, it doesn't do escaping of quotes. If you try searching for the phrase 'a "quoted" string' on the windows client, you get no songs found, and "invalid search request" printed in yellow in your console window. (don't know what code that is, sorry.)
- >
- > and no wonder-- a little birdie told me that the client sends this:
- >
- > FILENAME CONTAINS "a "quoted" string" MAX\_RESULTS 100

[contributed by Ben Byer <bbyer@rice.edu>. -ed]

Note that unlike the IRC protocol, each line does NOT end in \r\n. The <length> field specifies exactly how much data you should read.

### 3. Message Types

The following section describes the format of the <data> section for each specific message type. Each field is denoted with <>. The fields in a message are separated by a single space character (ASCII 32). Where appropriate, examples of the <data> section for each message are given.

<type> can be one of the following (converted to big-endian):

0 error message [SERVER]

Format: <message>

2 login [CLIENT]

Format: <nick> <password> <port> "<client-info>" <link-type> [ <num> ]

<port> is the port the client is listening on for data transfer. if this value is 0, it means that the client is behind a firewall and can only push files outward. it is expected that requests for downloads be made using the 500 message (see below)

<client-info> is a string containing the client version info

<link-type> is an integer indicating the client's bandwidth

- 0 unknown
- 1 14.4 kbps
- 2 28.8 kbps
- 3 33.6 kbps
- 4 56.7 kbps
- 5 64K ISDN
- 6 128K ISDN
- 7 Cable
- 8 DSL
- 9 T1
- 10 T3 or greater

<num> build number for the windows client [optional]

Example:

foo badpass 6699 "nap v0.8" 3

3 login ack [SERVER]

Format: <email>

the server sends this message to the client after a succesful login (2). If the nick is registered, the <email> address given at registration time is returned. If the nick is not registered, a dummy value is returned.

4 version check [CLIENT]

Format: <version>

Server sends [5] if an update is needed. All other responses are ignored.

<version> = string, version 2.0b5a sends "2.0"

## 5 "auto-upgrade" [SERVER]

Format: <version> <hostname:filename>

Napster is out of date, get a new version.  
Or also known as gaping security hole.

<version> = string, the new version number.  
<hostname> = string, hostname of upgrade (http) server  
<filename> = string, filename

Connections are always made to port 80.

The HTTP Request:

GET <filename> HTTP/1.0  
Connection: Keep-Alive  
Host: <hostname>

Expected HTTP Response.

Content-Length: <size>  
Content-Type: <doesn't matter>  
<data>

Upgrade file is save as "upgrade.exe".  
And executed as: upgrade.exe UPGRADE "<current.exe>"

As far as I can tell no confirmation is requested by Napster when it receives this message. And immediately start to "auto-upgrade". To keep users informed that Napster is doing something potentially very harmful to their computer it displays a message saying it's "auto-upgrading".

I think this pretty bad, since all someone has to do is hack a napster server et voila a zillion clients at your disposal.

As far as I known this only affects the Win32 2.0b5a Napster client. Other clients/versions I don't know.

[This section was contributed by Thomas van der Heijden <thom@bart.nl> -ed]

## 6 new user login [CLIENT]

Format: <nick> <pass> <port> "<client-info>" <speed> <email-address>

This message is used when logging in for the first time to register a new nickname. The client normally checks for an unused nick using the "check nickname" (7) message and upon receipt of a "nickname not registered" (8) from the server will proceed to log in with this command.

note: this message is similar to the 0x02 message, with the addition of <email-address> on the end

Example:

foo foo 6699 "nap v0.8" 3 email@here.com

## 7 nick check [CLIENT]

Format: <nick>

Queries the server to see if <nick> is already registered. This

message is typically used prior to logging in for the first time to check for a valid nickname.

Response to this message is one of 8, 9 or 10

8 nickname not registered [SERVER]

Format: (empty)

The server sends this message in response to the "nick check" (7) message to indicate that the specified nickname is not already registered and is ok to use.

9 nickname already registered [SERVER]

Format: (empty)

The server sends this message when the nickname the client has requested has already been registered by another user

10 (0x0a) invalid nickname [SERVER]

Format: (empty)

This server sends this message in response to the "check nickname" (7) messages when the specified nickname is invalid, usually due to bad characters such as spaces or other non-printable characters.

The Napster, Inc. servers only accept (upper- and lower-case) nicks comprised of the following:

abcdefghijklmnopqrstuvwxyz1234567890\_[]{}~@^!\$

11 ??? [CLIENT]

Format: <nick> <pass>

[returns "parameters are unparsable" -ed]

12 ??? [SERVER]

Format: (empty)

this message is returned in response to message 11 from the client

13 echo??? [SERVER]

Format: <numeric>: [args]

Prior to issuing a login (2) command, the server will echo back the received numeric and any arguments of any commands it receives.

14 login options [CLIENT]

NAME:%s ADDRESS:%s CITY:%s STATE:%s PHONE:%s AGE:%s INCOME:%s EDUCATION:

Example:

NAME: kev ADDRESS: CITY: ephrata STATE: pa PHONE: AGE: 60 or older

100 (0x64) client notification of shared file [CLIENT]

Format: "<filename>" <md5> <size> <bitrate> <frequency> <time>

<md5> see section "MD5"  
 <size> is bytes  
 <bitrate> is kbps  
 <frequency> is hz  
 <time> is seconds

Example:

"generic band - generic song.mp3" b92870e0d41bc8e698cf2f0a1ddfeac7.4433:

102 (0x66) remove file [CLIENT]

Format: <filename>

client requests to remove file from shared library

110 unshare all files [CLIENT]

Format: (empty)

Notifies the server that the client is no longer sharing any of the files previously shared.

200 (0xc8) client search request [CLIENT]

[FILENAME CONTAINS "artist name"] MAX\_RESULTS <max> [FILENAME CONTAINS "song"] [LINESPEED <compare> <link-type>] [BITRATE <compare> "<br>"] [FREQ <compare> "<freq>"] [WMA-FILE] [LOCAL\_ONLY]

The artist name and the song name are, obviously, treated the same by the server; confirm this for yourself on the windows client.

max is a number; if it is greater than 100, the server will only return 100 results.

<compare> is one of the following:  
 "AT LEAST" "AT BEST" "EQUAL TO"

<link-type> see 0x02 (client login) for a description

<br> is a number, in kbps

<freq> is a sample frequency, in Hz

LOCAL\_ONLY causes the server to only search for files from users on the same server rather than all linked servers.

The windows client filters by ping time inside the client. It pretty much has to, and it's easy to see the result by setting ping time to at best 100 ms or so, and max search terms to 50. You'll get back like 3 results, but the client will still tell you that it found "50 results".

Examples:

FILENAME CONTAINS "Sneaker Pimps" MAX\_RESULTS 75 FILENAME  
 CONTAINS "tesko suicide" BITRATE "AT LEAST" "128"

MAX\_RESULTS 100 FILENAME CONTAINS "Ventolin" LINESPEED  
 "EQUAL TO" 10

[Thanks to Ben Byer <bbyer@rice.edu> for this contribution. -ed]

## 201 (0xc9) search response [SERVER]

"<filename>" <md5> <size> <bitrate> <frequency> <length> <nick> <ip>  
<link-type> [weight]

<md5> see section "MD5"  
<size> is file size in bytes  
<bitrate> is mp3 bit rate in kbps  
<frequency> is sample rate in hz  
<length> is the play length of the mp3 in seconds  
<nick> the person sharing the file  
<ip> is an unsigned long integer representing the ip address of the user with this file  
<link-type> see message client login (2) message for a description  
[weight] a weighting factor to allow the client to sort the list of results. positive values indicate a "better" match, negative values indicate a "worse" match. If not present, the weight should be considered to be 0.

## Example:

"random band - random song.mp3" 7d733c1e7419674744768db71bffe8bcd 2558195

## 202 (0xca) end of search response from server [SERVER]

Format: (empty)

## 203 (0xcb) download request [CLIENT]

Format: <nick> "<filename>"

client requests to download <filename> from <nick>. client expects to make an outgoing connection to <nick> on their specified data port.

## Example:

mred "C:\Program Files\Napster\generic cowboy song.mp3"

SEE ALSO: 500 alternate download request

## 204 (0xcc) download ack [SERVER]

<nick> <ip> <port> "<filename>" <md5> <linespeed>

server sends this message in response to a 203 request.

<nick> is the user who has the file  
<ip> is an unsigned long integer representing the ip address  
<port> is the port <nick> is listening on  
<filename> is the file to retrieve  
<md5> is the md5 sum  
<linespeed> is the user's connection speed (see login(2))

## Example:

lefty 4877911892 6699 "generic band - generic song.mp3" 10fe9e623b1962d1

## 205 (0xcd) private message to/from another user [CLIENT, SERVER]

<nick> <message>



note the same type is used for a client sending a msg or recieving one

[Commentary: this message causes problems if you consider linking servers together. With the current one server situation, the server just rewrites the message it receives with the name of the client that sent it and passes it to the recipient client. However, in the case where the recipient and sender are not on the same server, there is loss of information without encapsulating it. It would have been better to put both the sender and recipient because if the servers are ever linked they will have to make a new message type for this situation. -ed]

206 (0xce) get error [SERVER]

<nick> "<filename>"

the server sends this message when the file that the user has requested to download is unavailable (such as the user is not logged in).

207 (0xcf) add hotlist entry [CLIENT]

<user>

This message is used to add additional entries to the client's hotlist. The server will send 209 and 210 messages when a user on the hotlist has logged in or out, respectively.

208 (0xd0) hotlist [CLIENT]

<user>

This message is used to send the initial list of hotlist entries during the initial login process. It is normally send prior to to the add file (100) commands. To add more entries to the hotlist after the initial list is sent, clients should use the 207 message instead.

209 (0xd1) user signon [SERVER]

<user> <speed>

server is notifying client that a user in their hotlist, <user>, has signed on the server with link <speed>

210 (0xd2) user signoff [SERVER]

<user>

server is notifying client that a user on their hotlist, <user>, has signed off the server.

this message is also sent by the server when the client attempts to browse a nonexistent client. [why don't they just use 404 for this? -ed]

211 (0xd3) browse a user's files [CLIENT]

<nick>

the client sends this message when it wants to get a list of the files shared by a specific client

7

## 212 (0xd4) browse response [SERVER]

<nick> "<filename>" <md5> <size> <bitrate> <frequency> <time>

<nick> is the user contributing the file  
 <filename> is the mp3 file contributed  
 <md5> is the has of the mp3 file  
 <size> is the file size in bytes  
 <bitrate> is the mp3 bitrate in kbps  
 <frequency> is the sampling frequency in Hz  
 <time> is the play time in seconds

Example:

foouser "generic band - generic song.mp3" b92870e0d41bc8e698cf2f0a1ddfe:

## 213 (0xd5) end of browse list [SERVER]

<nick> {ip}

indicates no more entries in the browse list for <user>. <ip> gives the client's IP address.

## 214 (0xd6) server stats [CLIENT, SERVER]

client: no data  
 server: <users> <# files> <size>

<size> is approximate total library size in gigabytes  
 this message is sent by the server occasionally without request

Example:

553 64692 254

## 215 (0xd7) request resume [CLIENT]

<checksum> <filesize>

client is requesting a list of all users which have the file with the characteristics. the server responds with a list of 216 messages for each match, followed by a 217 message to terminate the list

## 216 (0xd8) resume search response [SERVER]

<user> <ip> <port> <filename> <checksum> <size> <speed>

this message contains the matches for the resume request (215). the list is terminated by a 217 message.

## 217 (0xd9) end of resume search list [SERVER]

no data.

this message terminates a list of 216 messages initiated by a 215 client request

## 218 (0xda) downloading file [CLIENT]

no body.

client sends this message to the server to indicate they are in the

process of downloading a file. this adds 1 to the download count which the server maintains.

219 (0xdb) download complete [CLIENT]

no body.

client sends this message to the server to indicate they have completed the file for which a prior 218 message was sent. this subtracts one from the download count the server maintains

220 (0xdc) uploading file [CLIENT]

no body.

client sends this message to indicate they are uploading a file. this adds one to the upload count maintained by the server.

221 (0xdd) upload complete [CLIENT]

no body.

client sends this message when they are finished uploading a file. this subtracts one from the upload count maintained by the server.

300 (0x12c) optional ports [CLIENT]

<port>

This command allows the client to specify optional ports to try for data connections if the one currently in use is unreachable by other parties.

301 (0x12d) hotlist ack [SERVER]

<user>

server is notifying client that <user> has successfully be added to their hotlist

302 (0x12e) hotlist error [SERVER]

<user>

server is notifying client that it was unable to add <user> to their hotlist. [can you only add registered nicks to your hotlist? -ed]

303 (0x12f) remove user from hotlist [CLIENT]

<user>

client is notifying the server that it no longer wishes to request notifications about <user> when they sign on or off the server. no response is sent in return.

310 ??? [CLIENT, SERVER]

client: no data  
server: 0

unknown command. server returns 0 regardless of any parameters

316 client disconnect??? [CLIENT, SERVER]

client: no data  
server: 0

The server sends this message with a value of '0' when the client is about to be disconnected.

It is unknown what this command does when issued by the client. The server appears to send the 316 message without disconnected the client in this case.

[the server seems to send this message if you send it a numeric greater than 1000. the server will also disconnect you after sending this message. -ed]

320 (0x140) user ignore list [CLIENT, SERVER]

client: no data  
server: <count>

client request to display the list of ignored users.  
server returns the number of users being ignored

321 (0x141) user ignore list entry [SERVER]

<user>

server sends each ignored nick in response to a 320 request. the list is terminated by a 320 message with the number of ignored users.

322 (0x142) add user to ignore list [CLIENT, SERVER]

<user>

server acks the request by returning the nick

323 (0x143) remove user from ignore list [CLIENT]

<user>

server acks the request by returning the nick to be removed from the ignore list.

324 (0x144) user is not ignored [SERVER]

<user>

server indicates that <user> is not currently ignored in response to a 323 request.

325 (0x145) user is already ignored [SERVER]

<user>

server indicates the specified user is already on the user's ignore list

326 (0x146) clear ignore list [CLIENT, SERVER]

client: no data.  
server: <count>

client requests the server clear its ignore list. server returns the

Server

number of entries removed from the ignore list.

330 (0x14a) blocked ip list [CLIENT]

332 (0x14c) block ip [CLIENT]

333 (0x14d) unblock ip [CLIENT]

400 (0x190) join channel [CLIENT]

<channel-name>

the client sends this command to join a channel

401 (0x191) part channel [CLIENT, SERVER]

<channel-name>

The client sends this command to part a channel. Server sends this message to indicate that the client has parted the channel. Note that this is sometimes sent even when the client has not requested, so a client must be prepared to accept it at any time.

402 (0x192) send public message [CLIENT]

<channel> <message>

403 (0x193) public message [SERVER]

<channel> <nick> <text>

this message is sent by the server when a client sends a public message to a channel.

Example:

80's espinozaf hello...hola

404 (0x194) error message [SERVER]

<text>

This message is used to send general error messages to a client that is logged in. Note: Message 0 is only sent during the login process.

Examples:

User nosuchuser is not currently online.  
Channel #nosuchchannel does not exist!  
permission denied  
ping failed, blah is not online

405 (0x195) join acknowledge [SERVER]

<channel>

the server sends this message to the client to acknowledge that it has joined the requested channel (400)

406 (0x196) join message [SERVER]

<channel> <user> <sharing> <link-type>

17

<user> has joined <channel>

Example:

80's WilmaFlinstone 12 2

407 (0x197) user parted channel [SERVER]

<channel> <nick> <sharing> <linespeed>

Example:

80's DLongley 23 7

408 (0x198) channel user list entry [SERVER]

this message is identical to the join (406) message. the server will send the list of users in the channel prior to the client join command in this message. joins that occur after the client has joined will be noted by a 406 message.

409 (0x199) end of channel user list [SERVER]

<channel>

this message is sent by the server to indicate it has sent all informati

410 (0x19a) channel topic [CLIENT, SERVER]

<channel> <topic>

sent when joining a channel or a new topic is set. a client requesting topic change also uses this message.

[why didn't they put a field to indicate WHO changed the topic? as it is now you can only tell that it was changed. -ed]

420 (0x1a4) channel ban list [CLIENT, SERVER]

Format: <channel>

This command is used by clients to request the list of channel bans, and by the server to indicate the end of the channel ban list for a channel (also see 421).

421 (0x1a5) channel ban list entry [SERVER]

422 (0x1a6) channel ban [CLIENT]

<channel> <user|ip> [ "<reason>" ]

423 (0x1a7) channel unban [CLIENT]

<channel> <user|ip> [ "<reason>" ]

424 (0x1a8) channel ban clear [CLIENT]

Format: <channel>

425 channel motd (alternate topic (?)) [SERVER]

Format: <message>

12

[The server sends this command when a user joins one of the predefined channels. It seems to send a default message if the topic for the channel is the default, otherwise it sends the current channel topic. -ed]

430 invite a user [CLIENT, SERVER]  
 client: <nick> <channel>  
 server: <nick> <channel> "<topic>" <unknown\_digit> <unknown\_text>

This command is used by Napster 2.0 BETA 9.6 to invite a user to a channel.

client:  
     <nick> - nick of invited user  
     <channel> - channel user <nick> was invited to  
 server:  
     <nick> - nick of user who was inviting  
     <channel> - channel  
     <topic> - channel's topic  
     <unknown\_digit> - ??? ("0" works fine)  
     <unknown\_text> - ??? ("Hello!" works fine)

Last two arguments i cannot check because i temporary don't have internet access (e-mail only). If someone will test this message on napster.com servers please send me results.

When user receives 430 message user should reply with 431 or 432.

Client example:

CyberAlien2 #test\_channel

Server example:

CyberAlien3 #test\_channel "Welcome to the #test\_channel." 0 Hello!

!!! If anyone known what should server reply instead of "0 Hello!" please tell me.

431 invitation accepted [CLIENT]  
 <user> <channel>  
 <user> - user who was inviting

432 invitation declined [CLIENT]

format: copy of command received in 430 message

500 (0x1f4) alternate download request [CLIENT]

<nick> "<filename>"

requests that <nick> make an outgoing connection to the requesters client and send <filename>. this message is for use when the person sharing the file can only make an outgoing tcp connection because of firewalls blocking incoming messages. this message should be used to request files from users who have specified their data port as 0 in their login message

501 (0x1f5) alternate download ack [SERVER]

<nick> <ip> <port> "<filename>" <md5> <speed>

this message is sent to the uploader when their data port is set to 0 to indicate they are behind a firewall and need to push all data outware. the uploader is responsible for connecting to the downloader to transfer the file.

600 (0x258) request user's link speed [CLIENT]

<nick>

601 (0x259) link speed response [SERVER]

<nick> <linespeed>

602 (0x25a) ??? [CLIENT]

<?>

server returns a 404 with "\*gulp\* Drink milk, it does a body good!"

603 (0x25b) whois request [CLIENT]

<nick>

604 (0x25c) whois response [SERVER]

<nick> "<user-level>" <time> "<channels>" "<status>" <shared>  
<downloads> <uploads> <link-type> "<client-info>" [ <total downloads>  
<total\_uploads> <ip> <connecting port> <data port> <email> ]

<user-level> is one of "User", "Moderator", "Admin" or "Elite"

<time> is seconds this user has been connected

<channels> is the list of channels the client is a member of, each separated by a space (ASCII 32)

<status> is one of "Active", "Inactive" (offline) or "Remote" (on a different server)

<shared> is number of files user has available for download

<downloads> is the current number of downloads in progress

<uploads> is the current number of uploads in progress

<link-type> see 0x02 (client login) above

<client-info> see 0x02 (client login) above

The following fields are displayed for user level moderator and above:

<total uploads>

<total downloads>

<ip>

note: can be "unavailable"

<connecting port>

<data port>

<email>

note: can be unavailable

Example:

lefty "User" 1203 "80's" "Active" 0 0 0 3 "nap v0.8"

605 (0x25d) whowas response [SERVER]

<user> <level> <last-seen>

if the user listed in a 603 request is not currently online, the server sends this message.

11.



<user> is the user for which information was requested  
 <level> is the user's last known userlevel (user/mod/admin)  
 <last-seen> is the last time at which this user was seen, measured  
 as seconds since 12:00am on January 1, 1970 (UNIX time\_t).

606 (0x25e) change user level [CLIENT]

<nick> <level>

changes the privileges for <nick> to <level>. client must be admin level to execute this request

[I have not verified this message since I don't have admin status on any of the servers. -ed]

607 (0x25f) upload request [SERVER]

<nick> "<filename>" <speed>

this message is used to notify the client that user <nick> has requested upload of <filename>

Example:

lefty "generic band - generic song.mp3" 7

608 (0x260) accept upload request [CLIENT]

<nick> "<filename>"

client is notifying server that upload of <filename> to <nick> is accepted, and that the requesting client may begin download

Example:

lefty "generic band - generic song.mp3"

609 (0x261) accept failed [SERVER]

<nick> "<filename>"

this message is sent by the server when the client that the file was requested from does not accept the upload request.

610 (0x262) kill (disconnect) a user [CLIENT]

<nick> [ "<reason>" ]

client request to disconnect a user.

611 (0x263) nuke a user [CLIENT]

<nick>

client request to delete account for <nick>

612 (0x264) ban user [CLIENT]

<nick | ip> [ "<reason>" ]

client request to place a ban on either a specific nickname or an ip address

613 (0x265) set data port for user [CLIENT, SERVER]

client: <user> <port>  
server: <port>

This command is used by administrators to request that another client set the port used for data transfers to <port>. The server sends a message with the requested port number to the target client. NOTE: the target client can change its port number to whatever it wishes using the 703 command.

614 (0x266) unban user [CLIENT]

Format: <nick | ip> [ "<reason>" ]

615 (0x267) show bans for server [CLIENT]

Format: (empty)

client requests the list of banned ips for the current server

616 (0x268) (ip?) ban list entry [SERVER]

Format: <ip> <nick> "<reason>" <time> <n>

<ip> is the string version of the ip banned  
<nick> is the user setting the ban  
<reason> is the reason given  
<time> is the time\_t when the ban was set  
<n> is ??? value is either 0 or 1

This message is sent in response to the 615 client request, one for each ban. The list is terminated with a 0 length 615 message from the server.

Example:

207.172.245. valkyrie "DoS exploit" 947304224 0

617 (0x269) list channels [CLIENT, SERVER]

Format: (empty)

client requests a list of channels on the server. server responds with 618/617

server indicates end of channel list using this message.

618 (0x26a) channel list entry [SERVER]

Format: <channel-name> <number-of-users> <topic>

this is the server response to a 617 client request, one for each channel.

Example:

Help 50 OpenNap help channel

619 (0x26b) queue limit [CLIENT]

Format: <nick> "<filename>" <n>

a client may limit the number of downloads from a particular client. once the limit for a particular user has been reached, the uploading client can send this message to notify the downloader that they have hit the limit and can't have any more simultaneous downloads. <nick> is the user who hit the limit, <filename> is the file they were trying to download when they hit the limit, and <n> is the number of simultaneous downloads allowed.

Example:

joebob "C:\MP3\Generic Band - Generic Song.mp3" 3

620 (0x25c) queue limit [SERVER]

<nick> "<filename>" <filesize> <digit>

This message is sent by the server in response to the 619 client request, when one user needs to notify another that they have reached the maximum allowed simultaneous downloads. When the server receives the 619 request from the uploading client, it sends the 620 message to the downloading client. The only difference in format is the addition of the <nick> field in the 620 message which specifies the username of the uploading agent which is notifying the downloader that the queue is full.

Example:

joebob "C:\MP3\Generic Band - Generic Song.mp3" 1234567 3

621 (0x26d) message of the day [CLIENT, SERVER]

<text>

Server: each 621 message contains a single line of text

Client: client sends a 621 command with no data to request the motd be sent. The server will usually send this automatically after a user logs in, so this command allows a user to reread it upon request.

622 (0x26e) muzzle a user [CLIENT]

<nick> [ "<reason>" ]

client requests that <nick> not be allowed to send public messages

623 (0x26f) unmuzzle a user [CLIENT]

<nick> [ "<reason>" ]

client requests that the enforced silence on <nick> be lifted

624 (0x270) un-nuke a user [CLIENT]

<user>

625 (0x271) change a user's linespeed [CLIENT]

<user> <speed>

626 (0x272) data port error [CLIENT, SERVER]

<user>

When a downloading client detects that the uploader's data port is unreachable, it should send a 626 message to the server with the nick of the user for which the connection failed. The server then relays the message to the uploader, substituting the downloader's nickname in the message.

627 (0x273) operator message [CLIENT, SERVER]

client: <text>  
server: <nick> <text>

client request to send a message to all admins/moderators

628 (0x274) global message [CLIENT, SERVER]

client: <text>  
server: <nick> <text>

client request send a message to all users

629 (0x275) banned users [SERVER]

Format: <nick>

when displaying the ban list for the server, this message is used to indicate banned nicknames.

630-639 missing

640 direct browse request [CLIENT, SERVER]

Client: <nick>  
Server: <nick> [ip port]

Client: request files for <nick>  
Server: <nick> is requesting your files. optionally, <ip> and <port> are given if the client getting browsed is firewalled

This message is sent to initiate a direct client to client browsing of shared files.

See section 5.3.

641 direct browse accept [CLIENT, SERVER]

Client: <nick>  
Server: <nick> <ip> <port>

The client to be browsed sends this message back to the server to indicate it is willing to accept a direct browse from <nick>.

The server sends this message to the requestor of the browse to indicate where it should connect in order to do a direct browse from <nick>.

See section 5.3

642 direct browse error [SERVER]

<nick> "message"

Server sends this message in response to a 640 request if there is an error (eg. both clients are firewalled, or the browser is not sharing any files).

See section 5.3

643-649 missing

650-651 ??? [CLIENT]

permission denied.

652 (0x28c) cloak user [CLIENT]

sets the current user to "invisible"

653-699 missing.

700 change link speed [CLIENT]

<speed>

client is notifying server that its correct link speed is <speed>, in the range 0-10 (see the login message for details).

701 change user password [CLIENT]

<password>

client wishes to change their password

702 change email address [CLIENT]

<email address>

client wishes to change their email address

703 change data port [CLIENT]

<port>

client is changing the data port being listened on for file transfers

704-747 missing.

748 login attempt [SERVER]

the server sends this message to a logged in client when another client attempts to log in with the same nickname.

749 missing.

750 (0x2ee) server ping [CLIENT, SERVER]

server: none

client: <user>

Napster 2.0b5a sends the username in a response to a 750 from the server.

[server returns an empty 750 command in response. server ping? -ed]

751 (0x2ef) ping user [CLIENT, SERVER]  
    <user>  
    client is attempting to determine if <user>'s connection is alive

752 (0x2f0) pong response [CLIENT, SERVER]  
    <user>  
    this message is sent in response to the the 751 (PING) request

753 (0x2f1) change password for another user [CLIENT]  
    <user> <password> "<reason>"  
    allows an administrator to change the password for another user

754-769 missing.

770 ??? [CLIENT]  
    permission denied.

771 ??? [CLIENT]  
    permission denied.

772-799 missing.

800 (0x320) reload config [CLIENT]  
    <config variable>  
    resets configuration parameter to its default value

801 (0x321) server version [CLIENT]  
    no data.  
    client request's a server's version

802-804 missing.

805 ???  
    [returns permission denied. -ed]

810 (0x32a) set config [CLIENT]  
    <config string>  
    request a change in server configuration variables

811 (0x32b) ??? [CLIENT]  
    [returns permission denied. -ed]

820 (0x334) clear channel [CLIENT]  
    <channel>  
    remove all users from the specified channel

- 821 (0x335)      redirect client to another server [CLIENT, SERVER]
- client: <user> <server> <port>  
server: <server> <port>
- This command allows an administrator to redirect clients to another server.
- 822 (0x336)      cycle client [CLIENT, SERVER]
- client: <user> <metaserver>  
server: <metaserver>
- This commands allows an administrator to make a client restart the login process by first connecting to the metaserver (redirector) at <host>.
- 823 (0x337)      set channel level [CLIENT]
- <channel> <level>
- Sets <channel> such that users must be at least <level> in order to enter the channel
- 824 (0x338)      emote [CLIENT, SERVER]
- client: <channel> "<text>"  
server: <channel> <user> "<text>"
- A variation of the public message command to indicate an action by the user. Often implemented as the "/me" command in IRC clients.
- 825 (0x339)      user list entry [SERVER]
- <channel> <user> <files shared> <speed>
- an 825 message is sent for each user in the channel specified by the 830 message
- Example:
- Help testor3 0 3
- [This appears to be exactly the same format as the 408 message. -ed]
- 826 (0x33a)      channel limit [CLIENT]
- <channel> <limit>
- sets the maximum number of users that may enter a channel. <limit> is an integer value between 0 and 999999999. setting it higher than this results in the limit being set to -1. by default, created channels have a limit of 200. there appears to be no restriction on any channel member changing the limit, and no notification is given
- 827 (0x33b)      show all channels [CLIENT, SERVER]
- no data.
- client request to show all channels, including "hidden" channels. the server also sends this message to terminate

828 (0x33c) channel list [SERVER]

<channel> <users> <n1> <level> <limit> "<topic>"

the server sends a list of 828 commands, one for each channel, including "hidden" channels that don't show up in the normal channel list.

<level> is the minimum user level required for entry into a channel  
<limit> is the max number of users allowed in a channel

<n1> is either 0 or 1. seems to be 1 if the channel was user created, or 0 if a predefined channel???

829 (0x33d) kick user from channel [CLIENT]

<channel> <user> [ "<reason>" ]

830 (0x33e) list users in channel [CLIENT, SERVER]

<channel>

client requests a list of all users in <channel>. server responds with a 825 response for each user, followed by an 830 response with no data [why didn't they just use the 409 message? -ed]

831 (0x33f) global user list [CLIENT]

[returns permission denied. -ed]

832-869 missing.

870 add files by directory [CLIENT]

Format: "<directory>" "<file>" <md5> <size> <bitrate> <freq> <duration>  
[ ... ]

This command allows a client to share multiple files in the same directory as a shortcut to multiple 100 (share file) commands. <directory> is the path which should be prepended to all the given filenames in the rest of the command. <file> is the name of the file to share \*without\* its directory component. When other clients do a browse/share, the real path will be reported as <directory>/<file>.

The portion of this command after the <directory> may be repeated as many times as necessary for files in the same directory. NOTE: most servers will not accept commands of length longer than 2048 bytes so you still may need to break up the files into multiple commands if there are many files in a single directory.

871-899 missing.

900 connection test [SERVER]

<ip> <port> <data>

<ip> - string, ip address to connect to.  
<port> - integer, port to connect to  
<data> - string, data to send to server.

Try to connect to <ip> on <port> for atmost 1000 seconds. If the connection succeeds send the <data> to target.



[reported by Thomas van der Heijden <thom@bart.nl>]

901 listen test [SERVER]

<port> <timeout> <data>

<port> - integer, port to listen on  
 <timeout> - integer, time to wait for connection in seconds  
 <data> - string, data to send after connection has been made.

Listen on <port> for <timeout> seconds. If a connection arrives, return <data> to sender.

[reported by Thomas van der Heijden <thom@bart.nl>]

920 ??? [CLIENT]

Format: 1

This is sent by the BETA8 client prior to login. Purpose unknown.

### 3. MD5

It looks like the vast majority of the files are hashed using the first 299,008 bytes of the file. There have been some cases where the hash matches at 300,032 bytes, but no correlation has been drawn as to when that happens. The speculation at this point is that it might have to do with the existence of a ID3v2 tag, or perhaps the file was sampled at 48kHz...?

The current method seems to be: skip id3v2, seek to frame sync and hash.

Note: the linux nap client (versions 0.7 - 0.9) seem to hash exactly 300,000 bytes, which is NOT what the official windows client does.

More recent Napster clients seem to send the string WMA-FILE as the checksum instead of the actual hash value for .wma files.

### 5. Client-Client Protocol

File transfer occur directly between clients without passing through the server. There are four transfer modes, upload, download, firewalled upload, firewalled download. The normal method of transfer is that the client wishing to download a file makes a TCP connection to the client holding the file on their data port. However, in the case where the client sharing the file is behind a firewall, it is necessary for them to "push" the data by making a TCP connection to the downloader's data port.

#### 5.1 Normal Downloading

Regardless of which mode, the downloading client will first issue either a search(200) or browse(211) command to the server. This returns a list of files and information on the client sharing the file. To request a download, a get(203) request is sent to the server. The server will respond with a get ack(204) containing more detailed information.

This is the point at which the different methods diverge. If the 204 get ack says that the remote clients data port is 0, you must send a 500 request to the server requesting that the remote client send the data to you. In this case you wait for the remote client to connect to your own data port.

In the case where the sharing client is not firewalled, you make a TCP connection to the data port specified in the 204 message from the server. The remote client should accept the connection and immediately send one

23

ASCII char, '1' (ASCII 49). Once you read this char, you send a request for the file you wish to download. First send the string "GET" in a single packet, then send

```
<mynick> "<filename>" <offset>
```

where <mynick> is your napster user name, <filename> is the file you wish to download, and <offset> is the byte offset in the file to begin the transfer at (if you are downloading for the first time, and not resuming a prior transfer, you should use 0 to start at the beginning of the file).

The remote client will then return the file size, or an error message such as "INVALID REQUEST" or "FILE NOT SHARED". Note that the file size is not terminated in any special way, and the best way to figure out the size is to keep reading until you hit a character that is not a digit (it will usually be 0xff which is the start of the MP3 frame sync header, but if a ID3v2 tag is present it might look different). Immediately following the file size is where the data stream for the file begins.

Once the data transfer is initiated, the downloader should notify the server that they are downloading a file by sending the 218 message. Once the transfer is complete, you send a 219 message to indicate you have finished the download. Note that this is cumulative, so that if you are downloading multiple files, you send one 218/219 pair for EACH concurrent download--this is how the server knows how many transfers you have going on. Likewise, the uploader should send one 220 message for each upload, and one 221 when each upload has finished.

## 5.2 Firewallled Downloading

As described above, when the file needs to be pushed from a client behind a firewall, the downloader sends a 500 message to the server. This causes a 501 message to be sent to the uploader, which is similar to the 204 message for a normal download.

Once the uploader receives the 501 message from the server, they should make a TCP connection to the downloader's data port (given in the 501 message). Upon connection, the downloader's client will send one byte, the ASCII character '1'. The uploader should then send the string "SEND" in a single packet, and then the information:

```
<mynick> "<filename>" <size>
```

where <mynick> is the uploader's napster user name, <filename> is the file being sent, and <size> is the size of the file in bytes.

Upon receipt, the downloading client will either send the byte offset at which the transfer should start, or an error message such as "INVALID REQUEST". The byte offset should be sent as a single packet in plain ASCII digits. Just as with above in section 4.1, a 0 byte offset indicates the transfer should begin at the start of the file.

Each client should notify the server that they are uploading or downloading with the 218/219 (downloading) or 220/221 (uploading) command pairs (see section 4.1 for more detailed information).

## 5.3 Client to Client Browsing

Napster 2.0 BETA 8 adds a feature which allows direct client-to-client browsing of file lists. To request a browse, a client uses the

```
640 <nick>
```

command. The server then sends a

```
640 <requester>
```

to the client which is getting browsed with the nick of the client that is requesting the browse. If the client accepts the browse request, it sends back a

```
641 <requestor>
```

to the server with the nick of the client requesting the browse. The server then sends a  
641 <nick> <ip> <port>  
to the requesting client. In the case of an error, the server will send a 642 command in response to the 640 command.

The browsing client then makes a TCP connection to the remote client's data port. After getting the "l" character, the browsing client sends a  
GETLIST

At which point the remote client sends its nick followed by a linefeed (\n) by itself in a single packet (ie, one write() call)  
<nick> LF

followed by the list of files being shared (the format being the same as the data of the "share" command). Each line is terminated by a linefeed char. At the end of the list, an additional linefeed char is sent and then the client closes the connection.

In the case that the remote client is firewalled, the browse list will have to be pushed to the requesting client. The remote client makes a TCP connection to the requesting client, then sends  
SENDLIST <nick>\n  
followed by the list of files, as with the "GETLIST" command response.

#### 6. Where to get more help?

Join the napdev mailing list by sending email to [napdev-subscribe@onelist.com](mailto:napdev-subscribe@onelist.com) or by visiting the community page <http://www.onelist.com/community/napdev/>. This list is designed for open source napster developers to share information about the specification or applications.

#### 7. Acknowledgements

A big THANKS goes to the following people who contributed valuable information to this specification:

Ben Byer <bbyer@rice.edu>  
JT <jtraub@dragoncat.net>  
Evan Martin <eeym@u.washington.edu>  
Colten Edwards (aka panasync@efnet) <edwards@bitchx.dimension6.com>  
Thomas van der Heijden <thom@bart.nl>